Universidad Autónoma de Madrid
Escuela politécnica superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

CONTROL DE UN ROBOT BASADO EN ARDUINO MEDIANTE UN DISPOSITIVO MÓVIL ANDROID

Autor: Jonathan Patricio Yajamín Yajamín
Tutor: Luis Fernando Lago Fernandez

JUNIO 2017

Control de un robot basado en Arduino mediante un dispositivo móvil Android

Autor: Jonathan Patricio Yajamín Yajamín
Tutor: Luis Fernando Lago Fernandez

Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
JUNIO 2017

Resumen

Resumen

Actualmente vemos cómo va aumentando la tendencia y el interés en que los niños empiecen antes su aprendizaje sobre las Tecnologías de la Información y las Comunicaciones (TIC). Esto viene motivado por el aumento en la demanda de profesionales en este sector y la imposibilidad de atenderla. El objetivo de una temprana introducción es aumentar el número de profesionales en el sector tecnológico en el futuro, donde la demanda será aún mayor debido a la adaptación de las tecnologías en el mundo laboral. Teniendo esto en cuenta y con motivo de acercar y preparar a los niños, se han empezado a comercializar robots con propósito educativo.

El presente trabajo de fin de grado tiene relación con esta tendencia puesto que su objetivo es la construcción de un robot bípedo y la implementación de una aplicación móvil para controlarlo, proporcionando otra herramienta para el acercamiento de los niños a la tecnología. La aplicación será el método de comunicación entre el usuario y el robot, y con ésta se podrá controlar los servomotores a bajo nivel permitiendo al usuario un control directo sobre los servomotores del robot.

El robot está basado en la plataforma *Arduino* mientras que la aplicación tiene como objetivo el sistema operativo móvil *Android*. La comunicación entre la aplicación y el robot se realiza mediante el uso de la conectividad bluetooth, la cual se implementa haciendo uso de la API bluetooth propia del sistema.

Así mismo, se ha creado un mecanismo de recolección de información que permite analizar los patrones de movimiento introducidos por el usuario, mediante el uso de la aplicación, con la finalidad de generar patrones motores que puedan ser utilizados con posterioridad.

Dicho esto, se puede dividir el proyecto en tres partes: la construcción del robot, la implementación de la aplicación y la generación de patrones motores.

Palabras Clave

TIC, sector tecnológico, robot bípedo, aplicación móvil, Arduino, sistema operativo móvil, Android, bluetooth, patrones motores.

Abstract

Nowadays there is an increasing interest in making the children approach Information and Communication Technologies (ICTs) at early ages. This is motivated by the big demand for qualified workers in this sector, which is hardly covered by new professionals. The main purpose of an early introduction of ICTs is to increase the number of specialized professionals in the technological sector in the near future, when the demand for this kind of workers will be presumably greater than now. With this objective, educational robots have entered the scene.

In this context, this bachelor's thesis describes the creation of a biped robot and the implementation of a mobile application that allows to control it using a smartphone. The application is used to send low level motor commands that directly control the angular positions of the robot's servomotors, allowing for a precise manipulation of the robot.

The robot is based on the *Arduino* platform and the mobile application has been developed for the *Android* system. The communication between the application and the robot uses bluetooth connectivity, and is implemented using the Android Bluetooth API.

Additionally, the application permits to record all the motor commands introduced by the user in order to analyze and synthesize new motor patterns that can be used later.

The project can be divided into three parts: the robot construction, the application implementation and the motor patterns generation.

Key words

ICTs, technological sector, biped robot, mobile application, Arduino, mobile operating system, Android, bluetooth, API, motor patterns.

Agradecimientos

En primer lugar, y principalmente, quiero dar las gracias a mi madre. No solo porque me ha apoyado durante todo el camino que he recorrido hasta llegar a este punto, sino también por todos los sacrificios que ha hecho para que tenga una educación.

Quiero dar las gracias a amigos y familia que han estado presentes, apoyándome a conseguir mis objetivos y animándome cuando las cosas se torcían.

Por último, a mi tutor, Luis Lago, por la ayuda y la guía que me ha dado para llevar a cabo este proyecto.

Índice general

Índice de Figuras

Índice de Tablas

1 Introducción

1.1. Motivación del proyecto

Desde hace unos años se ha visto en aumento la tendencia de empezar una educación en las TIC a edades tempranas. Esto se debe a la creciente demanda en el sector tecnológico relacionada con la baja tasa de profesionales que trabajan en este sector [1]. Como consecuencia de esta tendencia se ha empezado a promover el uso de robots para enseñar a los más pequeños de la casa con el fin de prepararlos para un futuro en el que estén rodeados de tecnología.

Durante los últimos años, múltiples empresas han ido comercializando robots con fines educativos, como los robots *Zowi* de BQ, *Dash & Dot* de Vicens Vives, *mBot* de Makeblock y *Robo Wunderkid* [1]. Junto a estos se utilizan *lenguajes de programación visual* como *Scratch* o *Blockly* [2, 3] para la introducción de la programación de manera gráfica y fácil de entender y utilizar [4].

Los robots se introducen como juguetes que ayudan a los niños a desarrollar un amplio rango de destrezas como por ejemplo la creatividad, la abstracción en la forma de pensar y el razonamiento [2].

Dicho esto, el objetivo de este trabajo de fin de grado es la construcción de un robot

bípedo sencillo y la implementación de una aplicación móvil para poder controlarlo a bajo nivel, generando patrones motores. En otras palabras, crear un juguete con el cual los niños puedan seguir con su introducción en el ámbito de la tecnología.

A diferencia de otras aplicaciones en el mercado, la aplicación que se desarrollará controla al robot enviando datos que se utilizan directamente sobre el servomotor en lugar de utilizar funciones predefinidas para su movimiento. De esta forma se permite un control a más bajo nivel, dotando al usuario de plena libertad de movimiento.

Con este desarrollo se nos abre la posibilidad de descubrir nuevas formas de movimiento y reutilizarlas, puesto que la aplicación incorporará además un sistema de recogida de información de los patrones introducidos por el usuario, con la cual se podrá generar nuevos movimientos mediante su estudio y análisis.

1.2. Objetivos

El desarrollo de este trabajo de fin de grado tiene dos objetivos principales:

- La construcción de un robot bípedo y la implementación de una aplicación móvil que se utilizará para la comunicación y control directo de los servomotores del robot.
- A partir de la aplicación, recolectar información de los patrones de movimiento introducidos en la aplicación con el fin de diseñar nuevas formas de movimiento para el robot.

Para conseguir estos objetivos se han planteado los siguientes hitos:

- Construcción del robot bípedo controlado por una placa Arduino o compatible con Arduino.
- Implementación del programa Arduino que se cargará en el robot.
- Implementación de la aplicación móvil para el control del robot y el registro de patrones de movimientos.
- Análisis de los patrones de movimientos introducidos por el usuario en la aplicación.

- Abstracción de los patrones de movimiento para la generación de nuevas formas de movimiento.

1.3. Organización de la memoria

En esta sección se expondrá la organización que se seguirá en la redacción del presente documento. Este consta de los siguientes capítulos:

- **Introducción**. En este capítulo se hace una breve introducción al proyecto que se ha realizado, exponiendo la motivación de este y sus objetivos.
- **Estado del arte**. Este capítulo se centra en darle contexto al trabajo y recoger información de trabajos similares y en los que éste se ha basado.
- **Diseño**. En este capítulo se trata el diseño de las distintas partes que forman el proyecto, las cuales son un robot y una aplicación móvil.
- **Desarrollo**. En este capítulo se explica el desarrollo del proyecto dividiéndolo en el desarrollo del robot y en el de la aplicación móvil.
- **Pruebas y resultados**. En este capítulo se exponen las pruebas que se han realizado para validar el proyecto, así como los resultados de las mismas.
- **Conclusiones y trabajo futuro**. En éste último capítulo se muestran las conclusiones a las que se han llegado tras la finalización del proyecto, así como futuros trabajos que se podrían realizar para dar continuidad a este proyecto.

2

Estado del arte

El proyecto gira entorno a la construcción de un *robot bípedo* y su control por lo que en este capítulo se procede a hablar del estado actual en el que se encuentra este campo dentro de la robótica.

En primer lugar, se detallará brevemente la historia de los robots bípedos para posteriormente enlazarlo con su uso, junto con otros tipos de robots, en la robótica educativa.

2.1. Historia

El interés en la mecánica de la locomoción de las piernas se remonta muchos años atrás, el cual ha sido mencionado en textos de las antiguas civilizaciones griegas, indias, egipcias y chinas [5, 6]. Como ejemplos de este interés podemos encontrar la referencia de la creación de un robot con forma humana por uno de los dioses griegos en la *Ilíada* de Homero; el hallazgo de un robot mecánico en las pirámides de Egipto y el diseño; y posible construcción, de primer autómata antropomórfico, ideado por Leonardo da Vinci (1495-1497) al cual se le conoce como el *robot de Leonardo* [5].

Como muchas otras ramas de la ingeniería, los estudios sobre robots capaces de andar se vieron impulsados por las nuevas invenciones en mecanismos, la ciencia de los

materiales, la electrónica, los sistemas de control y los ordenadores realizadas después de la Segunda Guerra Mundial[5]. En particular, el estudio de la locomoción bípeda y su implementación en la robótica empezó en el año 1966 con el desarrollo del proyecto conocido como *Lower Limp Model WL-1* (figura 2.1), en la universidad Waseda, Tokio [6].

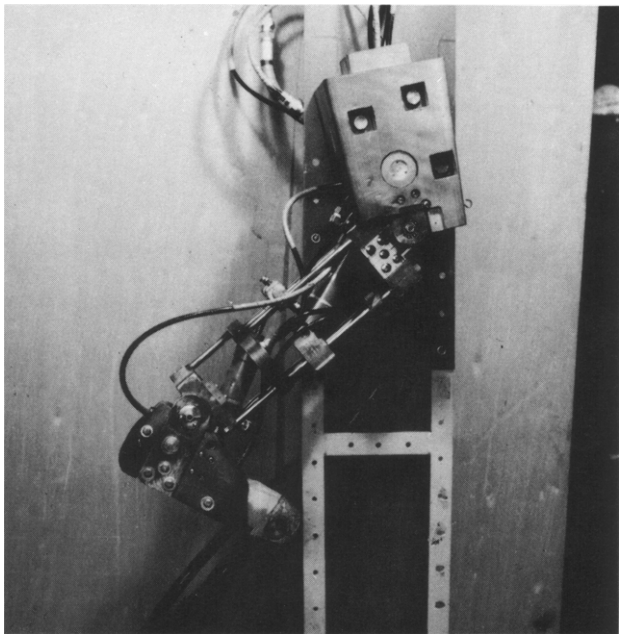

Figura 2.1: Lower Limp Model WL-1

Desde entonces, esta universidad ha dedicado muchos recursos al estudio y construcción de robots bípedos, convirtiéndola en una de las instituciones que más tiempo han dedicado al desarrollo de este tipo de robots [6]. Como resultados de estas investigaciones, en 1973, construyeron el robot **Wabot-I** que se convirtió en el primer robot antropomórfico a escala real del mundo [5, 6].

Esta universidad ha sido la protagonista de la consecución de más hitos dentro de la robótica bípeda como la construcción del primer robot capaz de andar realizando un movimiento cuasi-dinámico, el desarrollo de robots que podían descender ligeras pendientes y subir escaleras. A finales del siglo XX, en 1997, crearon el robot bípedo WABIAN, con el cual se buscaba que presentara pensamiento y patrones de comportamiento similares a los de los seres humanos [5].

Otro ejemplo dentro de Japón, es el desarrollo del robot **Honda Humanoid Robot**, el cual empezó en el año 1986 y concluyó en 1996 [5, 6]. Este robot estaba basado en las ideas de *inteligencia* y *movilidad* ya que su objetivo era coexistir, cooperar con seres humanos y ser usado en la vida diaria. En el año 2000, se construyó una versión más avanzada de este robot, el cual se conoce como **ASIMO**.

Japón es el país que más estudios ha realizado en relación a los robots humanoides [6] pero también podemos encontrar otros ejemplos de desarrollos de robots bípedos en otros países, algunos de estos son [5, 6]:

- La construcción de un exoesqueleto bípedo en el *Instituto Mihailo Pupin*, Belgrado.
- El desarrollo del robot humanoide **BHR-1** en la *Universidad de Ciencia e Ingeniería* de Beijing.
- La construcción del robot masculino **ARNE** y del robot femenino **ARNEA** en Rusia.
- La construcción de los robots **CB** y **DB** para el estudio de la locomoción bípeda por parte de la compañía *SARCOS*, Utah.

Durante los últimos años, el desarrollo de los robots bípedos ha ido aumentando y se ha expandido a ámbitos como el del entretenimiento o el aprendizaje. Un ejemplo de robot bípedo que se usa para el entretenimiento es el robot **Robi** (figura 2.2) de *Takahashi Tomotaka*.

Figura 2.2: Robi de Takahashi Tomotaka

2.2. Robótica educativa

La robótica educativa tiene como objetivo proveer un conjunto de experiencias que facilitan el aprendizaje de los estudiantes mediante la concepción, creación y puesta en marcha de prototipos robóticos y programas especializados con fines pedagógicos [3] independientemente de las orientaciones profesionales a las que se dediquen en el futuro [7].

La implementación de este tipo de enseñanza permite poner en práctica un amplio rango de conocimientos que se han aprendido en diversas asignaturas impartidas en el centro educativo. De esta forma, se motiva al estudiante al aprendizaje de estas materias ya que pueden utilizar el conocimiento teórico adquirido de manera práctica [3]. Esta motivación no es algo que se pueda aplicar solo a las asignaturas relacionadas con las ciencias e ingenierías. Con el enfoque correcto se puede lograr que el aprendizaje de asignaturas ajenas a este ámbito se vea beneficiado por la aplicación de la robótica educativa [7].

El aprendizaje de robótica y de programación potencia el pensamiento computacional, que es un proceso mental que se relaciona con la habilidad de resolver problemas complejos [8]. Además, ayuda al desarrollo de otras cualidades como la creatividad, el razonamiento lógico y la confianza en sí mismo [2, 3].

La expansión de la robótica educativa se debe a la aparición de la robótica de bajo coste, junto a la aparición y popularización de plataformas de hardware libre como la de *Arduino* o *Raspberry Pi* [9, 7] y la capacidad de imprimir las piezas que componen el robot utilizando impresoras 3D [8].

En el proceso de aprendizaje mediante la robótica se puede reconocer cuatro fases: imaginar, diseñar, construir y programar [7].

- **Imaginar**. Los estudiantes imaginan o debaten sobre qué ideas se pueden llevar a cabo, poniendo en acción su creatividad.

- **Diseñar**. Se concreta el proyecto que se desarrollará. Esto ayuda a la vinculación entre la imaginación y el mundo físico plasmando ideas y provoca la necesidad de utilizar conocimiento ya adquirido o de investigar.

- **Construir**. Se lleva a la práctica el conocimiento teórico. Permite la creatividad mediante el trabajo manual.

- **Programar**. Se programa el funcionamiento del robot, permitiendo el desarrollo del razonamiento lógico, la abstracción de ideas y el análisis espacial.

Actualmente se distribuyen numerosos robots con la educación como objetivo principal. Entre estos podemos encontrar el robot *Zowi* (figura 2.3) de BQ, *LEGO Education WeDo 2.0* de LEGO y *Dash & Dot* (figura 2.4) de Vicens Vives entre otros.

Figura 2.3: Robot Zowi de BQ

Figura 2.4: Robot Dash & Dot de Vicens Vives

2.3. Zowi

Zowi es un robot educativo creado por la empresa BQ compatible con Arduino. Su objetivo es el acercamiento de los niños al mundo de la tecnología [10] y que estos vayan superando diferentes niveles de complejidad en programación [11]. Este robot puede ser desmontado con facilidad para conocer cómo funciona y de que está hecho [10].

Cuenta con una aplicación con la cual se puede controlar el robot mediante acciones predefinidas para avanzar hacia adelante, hacia atrás y a los lados. También ofrece más opciones como la de poder jugar a repetir los movimientos que realiza el robot.

Este robot es programable mediante el lenguaje visual *Bitloq*, cuyo concepto es similar a *Scratch* y permite la programación por bloques y ver los efectos en Zowi de forma directa [12]. Con este lenguaje se puede ampliar las funcionalidades del robot.

Además, permite ser modificado internamente, cambiando los componentes que forman el robot.

3 Diseño

Este capítulo estará dividido en dos partes. La primera se centra en el diseño del robot y la segunda en el diseño y flujo de la aplicación móvil.

3.1. Robot

El diseño del robot montado para la realización de este proyecto está basado en un proyecto que sienta las bases del robot de BQ, *Zowi* [10], del cual se ha hablado en la sección 2.3 y utiliza como base la plataforma *Arduino*. Este proyecto se encuentra bajo licencia *Creative Commons Attribution-ShareAlike 4.0 International License* [13] y sus fuentes están disponibles en *GitHub* [14].

3.1.1. Arduino

Arduino es una plataforma electrónica de código abierto (hardware libre) que hace uso de un micro-controlador [15]. Esta plataforma está enfocada a ser fácil de utilizar y a la versatilidad.

Cuenta con un entorno de desarrollo (IDE) simple, el cual usa un lenguaje propio de programación basado en *Wiring*, el cual es un *framework* de código abierto para

micro-controladores [16]. Además de este lenguaje de programación, se pueden usar componentes programados en C++ [15].

Las placas Arduino estás formadas principalmente por un micro-controlador (usualmente AVR **Atmel**), pines de entrada y salida para conexiones tanto analógicas como digitales y memoria flash para almacenar el código a ejecutar [17].

La placa más popular es la *Arduino Uno* (figura C.1) que se incluye en numerosos kits de iniciación.

3.1.2. Componentes

Después de dar una breve introducción a la plataforma Arduino, se describirán los componentes que se usarán en la construcción del robot.

Piezas imprimibles

En primer lugar, se muestran las piezas imprimibles que se utilizan para dar forma al robot. Este está formado por dos pies, dos piernas, un chasis y un cuerpo (figura 3.1). Para la impresión de estas piezas se ha usado la impresora 3D de BQ, **Prusa i3 Hephestos**.

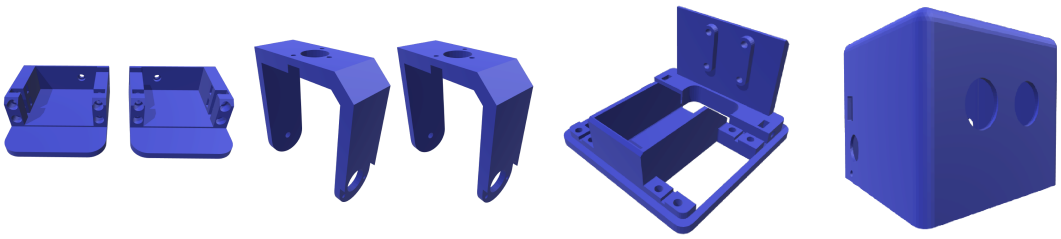

Figura 3.1: Piezas del robot

Servomotores

Un **servomotor** (también conocido como **servo**) es un tipo especial de motor el cual cuenta con un sistema de control. A diferencia de los motores de corriente continua, estos son más precisos, capaces de moverse hasta cualquier posición dentro de su rango de actuación y mantener su posición estable [18]. Un servomotor está compuesto por un

motor eléctrico, encargado de generar el movimiento; un sistema de regulación formado por engranajes, el cual actúa sobre el motor para regular su velocidad y fuerza que ejerce en su eje (*torque*); un circuito electrónico que actúa como sistema de control mediante el envío de señales moduladas por ancho de pulso (PWM); y un potenciómetro que indica la posición del eje en todo momento [19] (figura C.2).

Para este proyecto utilizaremos *servomotores de modelismo*, que son los utilizados frecuentemente en sistemas de radio-control y robótica. Estos operan con un voltaje reducido, comprendido entre los 4 y 6 voltios. Dentro de este tipo de motores podemos encontrar servomotores de rango limitado, capaces de girar 180^{o}, y los servomotores de rotación continua, capaces de realizar un giro completo (360^{o}) [20].

Los servomotores cuentan con tres terminales de conexión: dos para la alimentación (VCC y GND) y otro para la señal de control. En la figura C.3 se pueden ver los colores que suelen utilizarse para estos terminales.

Para la construcción del robot utilizaremos 4 servomotores Futaba S3003 (figura C.4), cuyas especificaciones son las siguientes:

Especificación	Valor
Velocidad	0.23 seg/60 grados (260 grados/seg)
Torque de salida	3.2 Kg-cm (0.314 N.m)
Dimensiones	40.4 x 19.8 x 36 mm
Peso	37.2 gramos
Frecuencia de PWM	50Hz (20ms)
Rango de giro	180 grados

Cuadro 3.1: Especificaciones del servomotor Futaba S3003

Micro-controlador

Al principio de la realización de este proyecto, se experimentó con la placa Arduino Uno, de la cual se ha hablado en la sección 3.1.1, junto con el módulo bluetooth HC-05 para poder realizar la conexión con el dispositivo móvil. Para poder utilizar el módulo bluetooth era necesario el uso de una *protoboard* para realizar la conexión placa-módulo.

Posteriormente, se cambió la placa Arduino Uno por la placa **BQ ZUM BT 328** (figura C.5), la cual es compatible con Arduino. El cambio se debe a que esta cuenta con un módulo bluetooth incorporado y posee pines de conexión en los que conectar

los terminales de los servos directamente, sin necesidad de usar una *protoboard* para alimentar a los distintos servos. Esta placa tiene las siguientes características:

Especificaciones	Valor
Voltaje de entrada	5.8V – 16V
Corriente de salida	3.3V (50mA) – 5V (3.2A)
Velocidad de la CPU	16 MIPS
Pines digitales	14 I/O
Pines analógicos	6
Interfaces	Bluetooth 2.1 (Baud Rate 19200 bps), USB, TTL UART, SPI, I2C, ICSP

Cuadro 3.2: Especificaciones de la placa BQ ZUM BT 328

Alimentación

Para la alimentación del robot se utilizará un soporte para pilas de tipo AAA, el cuál necesita 8 pilas para proveer energía.

Tornillería

Los tornillos y tuercas que se usarán para unir las piezas imprimibles y componentes son los siguientes:

- 6 tornillos M3x10 (ISO 4762)
- 2 tornillos M3x10 (ISO 10642)
- 8 tornillos M3x12 (ISO 4762)
- 8 tornillos M3x16 (ISO 4762)
- 22 tuercas M3 (ISO 4032)

3.1.3. Montaje

En esta sección se detalla el montaje del robot. Para él se ha utilizado el siguiente esquemático (figura 3.2), que muestra las conexiones hechas entre los componentes y el micro-controlador.

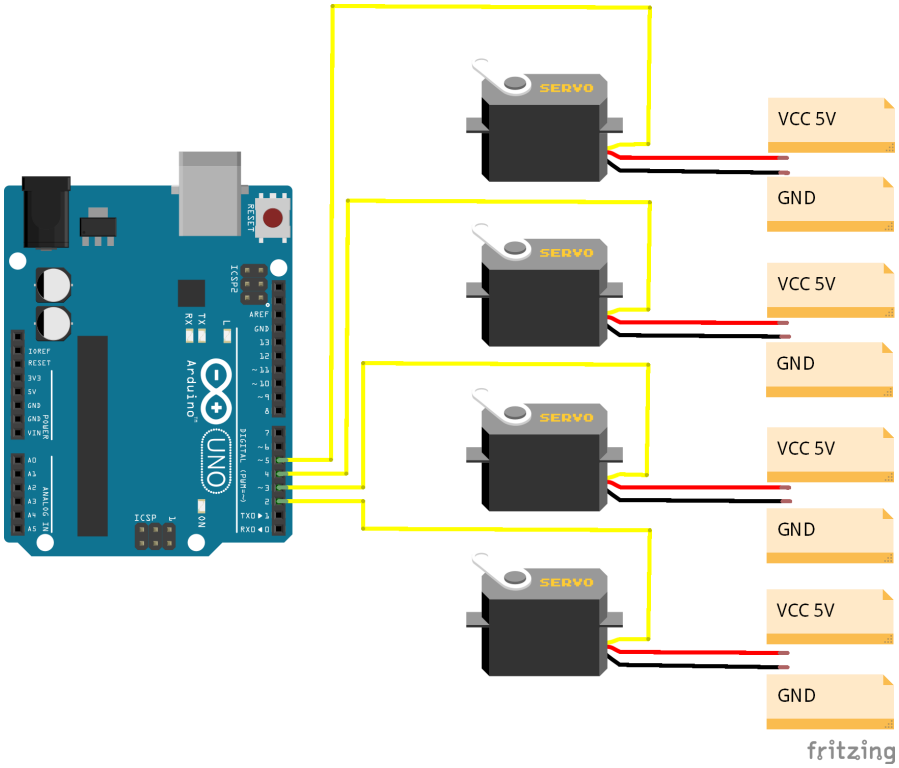

Figura 3.2: Esquemático del montaje del robot

A pesar de que en el esquemático se muestra la placa Arduino Uno, en el montaje real se ha utilizado la placa BQ ZUM BT 328, la cual posee una hilera de tres pines (VCC, GND y señal de control) en los que se conectan directamente los terminales del servomotor.

Como se puede ver en la imagen, hay cuatro servomotores conectados a los pines del 2 al 5. En orden creciente estos pines controlan los servos del lado izquierdo y derecho de la cadera y del pie izquierdo y derecho respectivamente. Teniendo esto en cuenta, se realizan los siguientes pasos para montar el robot.

1. Colocar los 2 tornillos M3x10 de cabeza cónica en la abertura que se encuentra en la parte de atrás de cada pie para proveer al servomotor de un doble eje.

2. Colocar 8 tuercas (4 en cada pie) en las aberturas interiores laterales de los pies para más adelante sujetar los servomotores que se encargarán del movimiento de los pies.

3. Con los servomotores incrustados en los pies, colocar los 8 tornillos M3x16 (4 en cada pie) en las aberturas frontales de los pies.

4. Con los servomotores posicionados en el chasis, introducir los 8 tornillos M3x12 (4 en cada lado) en las aberturas inferiores de este y usar una tuerca para fijar los servomotores al chasis.

5. Con la placa (BQ Zum BT 328) colocada en el respaldo del chasis, introducir 4 tornillos M3x10 por la parte frontal de la placa y enroscar una tuerca en los tornillos para fijar la placa al respaldo.

6. Realizar las conexiones que se muestran en el esquemático.

7. Colocar las dos tuercas restantes en las hendiduras laterales situadas delante del respaldo.

8. Por último, con el cuerpo cubriendo el chasis, enroscar los dos tornillos M3x10 restantes en las aberturas laterales del cuerpo del robot.

El resultado final del montaje se puede ver en la figura 3.3.

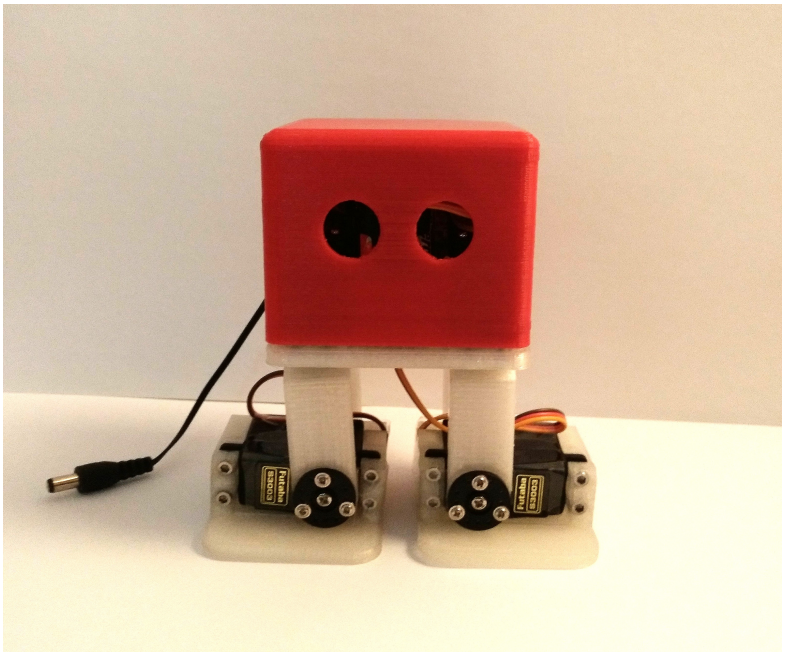

Figura 3.3: Robot construido

3.2. Aplicación móvil

En este apartado se describirá el diseño de la interfaz de usuario de la aplicación, así como los patrones de diseño utilizados en el desarrollo de la aplicación.

En el diseño de la interfaz se utilizará el nuevo patrón de diseño que ha venido utilizando Google desde el lanzamiento de *Android Lollipop*, *Material design* [21]. Este es un lenguaje visual para los usuarios, el cual sintetiza los principios de buen diseño con la innovación y la posibilidad de la tecnología y la ciencia [22]. Está basado en el uso de objetos materiales que se guían por leyes físicas y usan animaciones lógicas [21].

Para el diseño de la aplicación se ha usado de forma recurrente la guía de diseño de *material design* [22].

3.2.1. Flujo

Antes de pasar al diseño de las vistas de la aplicación, primero se expondrá el flujo que seguirá la aplicación (figura D.1).

Al iniciar la aplicación se mostrará una pantalla de inicio mientras se carga la aplicación (*splash screen*). Cuando el sistema haya cargado la aplicación, se presentan tres casos.

- Si es la primera vez que se abre la aplicación, se muestra una breve introducción solicitando la activación de los permisos pertinentes o de la activación de recursos necesarios.
- Si no es la primera vez y los permisos de localización se han revocado o la funcionalidad bluetooth esta desactivada, se muestran notificaciones pidiendo la concesión de permisos o la activación de la conectividad bluetooth. Una vez los permisos se han concedido y se ha activado el bluetooth, se muestra la vista de la que se habla en el siguiente punto.
- Si no es la primera vez, se han concedido los permisos y el bluetooth está activado, se muestra una pantalla que contiene dos listados; uno para los dispositivos emparejados con el dispositivo móvil y el otro para los dispositivos bluetooth disponibles alrededor.

Al seleccionar un dispositivo disponible en el listado, se procede al establecimiento de la conexión bluetooth con el robot. En caso de poder conectarse, la vista cambia y se muestra un menú general, en caso contrario se muestra un mensaje de error al usuario.

En el menú se muestran las siguientes opciones:

- Calibrar
- Controlador
- Grabar
- Grabaciones

De las cuatro opciones anteriores, la última es la que tiene un flujo distinto al de las demás. En esta se muestra un listado de las grabaciones que se han hecho y se permite reproducirlas, cambiar su nombre o eliminarlas. Al reproducir la grabación se cambia de vista y se muestran los datos de la grabación.

Una vez se ha establecido la conexión, si esta se pierde se redirigirá a la primera vista de la aplicación, el listado de dispositivos bluetooth.

3.2.2. Vistas

Una vez establecido el flujo que seguirá la aplicación, se definirá y mostrará el diseño de cada una de las vistas que componen la aplicación.

La aplicación cuenta con 12 vistas repartidas en las siguientes pantallas.

- Splash screen (figura E.1).
- Introducción (figura E.2).
- Listado de dispositivos bluetooth (figura E.3).
- Notificación de concesión o activación de permisos (figura E.4).
- Menú (figura E.5).
- Calibración (figura E.6).

- Controlador y grabación (figura E.7).
- Listado de grabaciones (figura E.8).
- Reproducción de la grabación (figura E.9).

Splash screen

Esta vista muestra el logo de la aplicación mientras el sistema carga la aplicación. Se utiliza para evitar que se muestre una pantalla en blanco durante el tiempo de carga.

Introducción

En el caso de la introducción, se muestran dos vistas. La primera informa de la necesidad de acceder a recursos de localización y en la que se solicitan los permisos para su utilización. La segunda notifica el uso de la conectividad bluetooth y solicita la activación en caso de que esta no este activada.

Dispositivos bluetooth

En esta vista se muestran dos listados: uno para los dispositivos emparejados y otro para los dispositivos disponibles. Cada elemento de la lista está formado por un icono, el nombre y la dirección MAC del dispositivo. Las listas pueden ser actualizadas realizando un movimiento *swipe down* (arrastrar hacia abajo).

Notificaciones

Las notificaciones de revocación de permisos de localización y desactivación de la conectividad bluetooth informan al usuario de la necesidad de estas funcionalidades. En caso de querer conceder los permisos o activar el bluetooth, el usuario cuenta con un botón para tal acción.

Menú

El menú está compuesto por cuatro botones que representan las opciones que se muestran en la sección 3.2.1: calibrar, controlador, grabar y grabaciones.

Calibración

Esta vista se utiliza para corregir el desfase en el valor de los ángulos usados en la inicialización de los servomotores dejando, de esta forma, al robot en un estado estable, con los pies formando un ángulo de 90º con las piernas. Estas al mismo tiempo han de estar en paralelo la una con la otra y paralelas a los laterales del robot. Además, la vista se usa para establecer la intensidad de los controles para mover el robot.

Para cada una de las dos opciones anteriores se muestran cuatro componentes *SeekBar* (uno por cada servomotor usado) que se utilizan para variar el valor de los ángulos y la intensidad.

En la parte inferior de la pantalla, y de manera fija, se muestran dos botones. El primero se utiliza para enviar los valores de los ángulos para realizar la corrección del desfase de los servomotores. El segundo se utiliza para guardar la configuración en las preferencias de la aplicación.

Controlador y grabación

A pesar de ser dos opciones diferentes en el menú, el diseño de estas vistas es el mismo. La única diferencia que hay entre ambas es la funcionalidad que realizan. Mientras en la primera solo se envían los comandos al robot, en la segunda además se guardan estos comandos en un fichero para su posterior reproducción.

La vista cuenta con dos controles *joystick* modificados, de forma cuadrada, para usarlos con el fin de mover los servomotores y con dos cuadros de texto encima de cada *joystick* para ofrecer *feedback* al usuario sobre el ángulo que se ha conseguido con el movimiento de los controles y que se usarán en el comando enviado.

Grabaciones

Esta vista muestra un listado de las grabaciones realizadas. En cada elemento de la vista se muestra el nombre de la grabación y la fecha de modificación. Además, cuentan con dos opciones: editar el nombre de la grabación y eliminarla. En caso de eliminar una grabación, se muestra una notificación (*Snackbar*) de confirmación, la cual cuenta con la opción de deshacer la eliminación.

Reproducción

En esta vista se muestran los comandos que se han utilizado en la grabación. Al igual que la vista de calibración, en la parte inferior hay una barra con la cual se puede pausar, reanudar, parar e iniciar la grabación.

3.2.3. Patrones de diseño

Un factor importante a tener en cuenta es los patrones de diseño que se utilizarán en el desarrollo de la aplicación. Estos son:

- Patrón **MVP** (Model-View-Presenter) para facilitar la testabilidad de la aplicación [23], mediante la separación de la interfaz de usuario de la lógica de la aplicación [24]. En el desarrollo de la aplicación se utilizará la versión de MVP de *vista pasiva* (figura 3.4).

- Patrón **singleton** (figura 3.5) para garantizar el uso de una sola instancia de un objeto a lo largo de la ejecución de la aplicación, proporcionando un punto de acceso global [25].

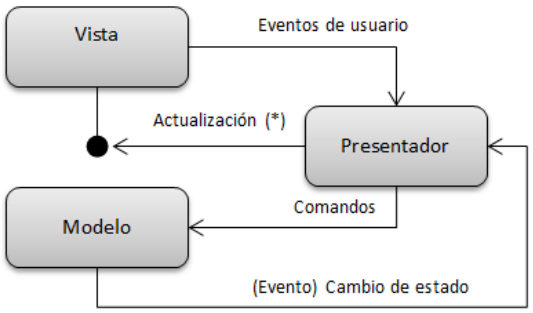

Figura 3.4: Patrón MVP como vista pasiva

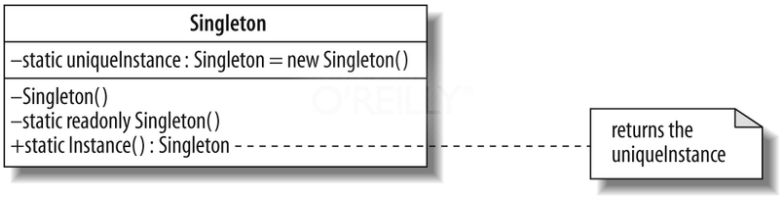

Figura 3.5: Patrón singleton

4 Desarrollo

Este capítulo, al igual que el anterior, se dividirá en dos partes. Por un lado, se tratará el desarrollo del código que se carga en el micro-controlador y por otro el desarrollo de la aplicación móvil.

4.1. Robot

Para el desarrollo y carga del software en el robot se ha usado el IDE de *Arduino*.

Como se ha dicho previamente, al principio de la realización de este proyecto se llevaron a cabo experimentos con los distintos componentes que forman el robot. Se implementaron varios *sketches*, programas Arduino, para aprender el funcionamiento de los servos, así como el uso del módulo bluetooth HC-05. También se creó un *sketch* para realizar una conexión bluetooth con una aplicación de terceros y con esta controlar el movimiento de un solo servomotor. Estos primeros experimentos se realizaron usando una placa Arduino Uno.

Una vez se tuvo más conocimiento sobre el entorno y sobre cómo utilizar los componentes del robot, se comenzó con el desarrollo del software final que da vida al robot. El proyecto Arduino principal consta de un solo fichero, el cual hace uso de la librería

Servo.h para el control de los servomotores.

Como es común en los programas que se cargan en una placa Arduino, el programa tiene una sección de código en la que se configuran los recursos que se van a utilizar (función *setup*) y otra en la que se realiza la acción (función *loop*), la cual se repite periódicamente hasta el final de ejecución, es decir, la desconexión del robot de la fuente de alimentación. El código se encuentra disponible en el anexo B.

En la sección de configuración se inicializa el ratio de transmisión (*baudios*) de la conexión serie. Esta se inicia con 19200 baudios, tal y como consta en las especificaciones del micro-controlador usado, la placa BQ ZUM BT 328.

La acción que se realiza en la función *loop* no es otra que la de decodificar los comandos que se envían a través de la aplicación móvil y realizar la acción correspondiente. Hay dos tipos de comandos que recibe el micro-controlador: comando de calibración y comando de control. Estos comandos están formados por pares letra-número separados por comas y terminados por punto y coma para marcar su finalización.

El comando de calibración se utiliza para corregir el desfase en el valor de los ángulos usados en la inicialización de los servomotores. Está compuesto por cuatro pares letra-número donde cada letra se corresponde a un servomotor.

Letra	Servomotor
A	Lado izquierdo de la cadera
B	Pie izquierdo
C	Lado derecho de la cadera
D	Pie derecho

Cuadro 4.1: Composición del comando de calibración

Ejemplo de comando de calibración. A5,B5,C0,D0;

El comando de control se utiliza para mover dos servomotores al mismo tiempo, puesto que se han agrupado en dos grupos: los servomotores del lado izquierdo y los del lado derecho. El comando está compuesto por dos pares letra-número donde cada letra se corresponde al lado que se quiere controlar y los números al valor de los ángulos de la cadera y el pie respectivamente.

Letra	Grupo
L	Izquierdo
R	Derecho

Cuadro 4.2: Composición del comando de control

Ejemplo de comando de control. L90,60;

Antes de empezar con la construcción del comando, se establece uno de los modos disponibles: calibración o control. El fin del establecimiento de uno de estos modos es que el programa sepa cómo tratar el comando que está leyendo, puesto que los dos tipos que hay son distintos y aportan distinta información.

```
void loop() {
    while(Serial.available()){
        letter = Serial.read();
        letterValue = (unsigned int)letter;

        if(letter == CONNECTION_NOTIFICATION){
            Serial.println("Connection accomplished");
        }
        else if(letter == CALIBRATION_MODE_NOTIFICATION){
            Serial.println("CalibrationAction mode activated");
            executionMode = Calibration;
        }
        else if(letter == CONTROL_MODE_NOTIFICATION){
            Serial.println("Control mode activated");
            executionMode = Control;
        }
        else if(letterValue != LINE_FEED && letterValue != CARRIAGE_RETURN)
    {
            if(executionMode == Calibration)
                calibrateRobot();
            else if(executionMode == Control)
                moveRobot();
        }
    }
}
```

Receiver.ino

Una vez se llega al final del comando, marcado por un punto y coma, se vacía la variable que contiene el comando (para preparar la captura del siguiente comando) y se ejecuta la acción que se ha especificado, moviendo los servomotores para calibrar el robot o para reflejar el movimiento hecho en la pantalla de control, grabación o reproducción de la aplicación.

4.2. Aplicación móvil

Como se ha dicho anteriormente, la aplicación que se ha desarrollado se ejecutará sobre la plataforma *Android*. Para su desarrollo se ha usado el IDE *Android Studio*.

Antes de exponer el desarrollo de la aplicación, primero se hablará de las librerías y recursos externos utilizados, permisos de la aplicación y los servicios de Android.

4.2.1. Librerías y recursos externos

En primer lugar, se nombrará las librerías y recursos que se han usado en el desarrollo de la aplicación, ajenas a las proporcionadas por la APIs de Android.

Se ha utilizado la librería *Butter Knife* [26] para utilizar inyección de dependencia sobre las vistas de los layouts. Esta librería encuentra y enlaza automáticamente la vista en el código proporcionando tan solo el nombre con el que se identifica la vista. Además de esta funcionalidad también es capaz de enlazar eventos, como *OnClick*, a funciones. Esta librería ayuda ahorrar líneas de código que muchas veces son repetitivas.

Por otro lado, la implementación de la vista del *joystick* virtual utilizado para el control del servomotor se basa en un proyecto de *GitHub* [27] el cual está bajo licencia *Apache License, Version 2.0* [28].

4.2.2. Permisos

A la hora de desarrollar una aplicación para Android son muy importantes los permisos que se solicitan y se conceden a la misma. Estos permisos se dividen en varios niveles de protección, pero los niveles más importantes son los permisos *normales* y los *riesgosos* [29].

Los permisos normales engloban la solicitud de información del usuario cuya concesión conlleva un riesgo mínimo para la privacidad del mismo o el correcto funcionamiento de otras aplicaciones. Si una aplicación declara que necesita un permiso de este tipo, en el fichero *AndroidManifest.xml*, el sistema se lo concede automáticamente sin la intervención del usuario.

Los permisos riesgosos abarcan áreas en las que la aplicación solicita información

privada del usuario o que podría afectar a datos almacenados del usuario o el funcionamiento de otras aplicaciones. A diferencia de los permisos normales, la concesión de los permisos riesgosos se hace de manera explícita. Hasta el nivel API 22 (*Lollipop 5.1*) se preguntaba al usuario si concedía estos permisos o no en el momento de la instalación de la aplicación, pero a partir de *Android Marshmallow* (API 23) este paradigma cambió; la solicitud de concesión de permisos se hace en tiempo de ejecución, cuando la aplicación necesita acceder a un recurso. De esta manera, la solicitud de permisos recae de forma completa en el desarrollador.

Los permisos que se van a utilizar en esta aplicación son los que acceden a los recursos de conectividad *bluetooth* y *localización*.

- **BLUETOOTH**: utilizado para solicitar y establecer la conexión y la transferencia de datos.
- **BLUETOOTH_ADMIN**: utilizado para la detección de dispositivos o control de ajustes del bluetooth.
- **ACCESS_COARSE_LOCATION**: utilizado para permitir acceso al proveedor *NETWORK_PROVIDER*. Es necesario para mejorar la protección de datos a partir del nivel de API 23.

A pesar de que la aplicación solo hace uso de recursos para establecer una conexión bluetooth, a partir de *Android Marshmallow* es necesario definir permisos de localización para poder buscar dispositivos bluetooth cercanos. Este cambio se introdujo para mejorar la protección de datos del usuario, abstrayendo y ocultando la dirección MAC del dispositivo que empieza el escaneo bluetooth. Durante el escaneo, la operación es visible para los dispositivos externos como si proviniera de una dirección MAC aleatoria [30].

Para definir los permisos necesarios en la aplicación, se ha añadido las siguientes líneas en el fichero *manifest*.

```
<uses-feature
        android:name="android.hardware.bluetooth"
        android:required="true" />
<uses-feature android:name="android.hardware.location.gps" />

<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

```
<uses-permission-sdk-23 android:name="android.permission.
    ACCESS_COARSE_LOCATION" />
```

AndroidManifest.xml

Además de los permisos, también se han declarado los recursos hardware que se requieren, bluetooth y localización, mediante el tag XML *uses-feature*. El objetivo de estas declaraciones es la de informar a una entidad externa sobre el conjunto requerimientos hardware y software de la aplicación [31]. En este caso se establece el requerimiento del hardware bluetooth como *required* puesto que sin este la aplicación no podría funcionar.

4.2.3. Servicios

Un servicio Android es un componente que puede realizar operaciones de larga duración en segundo plano. Este componente se seguirá ejecutando en segundo plano a pesar de que se cambie de aplicación [32].

Debido a su cualidad de realizar operaciones de larga duración, es el componente idóneo para mantener la conexión bluetooth entre el robot y el dispositivo móvil. El servicio que se ha implementado, se encarga del establecimiento de la conexión, así como del envío de datos a través del canal RFCOMM abierto.

4.2.4. API Bluetooth

La conectividad bluetooth es la característica más importante del desarrollo de este proyecto. Para establecer dicha conexión se utiliza la API bluetooth que se incluye en la plataforma Android, la cual permite el intercambio inalámbrico de datos con otros dispositivos.

Se ha utilizado la Android Bluetooth API para:

- Activar el recurso bluetooth en el dispositivo móvil.
- Buscar dispositivos bluetooth.
- Consultar al adaptador por los dispositivos sincronizados.
- Conectarse a otro dispositivo, estableciendo un canal RFCOMM.
- Transferir datos entre el dispositivo móvil y el robot.

4.2.5. Especificaciones del desarrollo

En esta sección se detallará el desarrollo de la aplicación centrando la atención en la estructura del proyecto no en el código fuente, cuyos fragmentos más relevantes se mostrarán en el anexo B.

Como se ha dicho en la sección 3.2.3, se ha usado el patrón **MVP** a lo largo de todo el desarrollo de la aplicación. Este patrón consiste en el desacoplamiento de las acciones realizadas sobre en la pantalla mediante el uso de interfaces. Por cada vista se han definido dos interfaces, una para las acciones que realiza el usuario sobre la interfaz gráfica y la otra para mostrar cambios en ésta, derivadas de las acciones del usuario. La vista, *actividad* o *fragmento*, es la que implementa la segunda interfaz mientras que para la implementación de la primera se crea una clase, conocida como *presentador*. Esta hace la acción de controlador supervisor y se encarga del manejo de la lógica de la aplicación.

Otra característica del desarrollo de la aplicación es la separación que se ha hecho del código. Este ha sido dividido de tal forma que cada paquete engloba una funcionalidad de la aplicación, haciendo que cada una sea independiente del resto y mejorando la legibilidad. Teniendo en cuenta esto, el proyecto ha sido dividido en las funcionalidades que se muestran en la siguiente tabla.

Paquete	Descripción
.calibration	Calibra el robot.
.connection	Se conecta al robot.
.controller	Controla al robot.
.intro	Introducción de la aplicación
.menu	Menú general
.play	Reproduce una grabación
.record	Graba los movimientos del robot.
.recordings	Muestra las grabaciones realizadas.
.splash	Muestra una vista mientras se carga la aplicación.
.model	Contiene los modelos de dominio de la aplicación.
.util	Contiene clases que se han utilizado en varias partes de la aplicación.

Cuadro 4.3: Estructura dentro del paquete *es.uam.eps.tfg.controller*

Dejando de lado la estructura del proyecto, se detallarán las funcionalidades de la aplicación y su implementación.

Introducción

Esta vista está compuesta por un *slider* (componente *ViewPager*), en el cual se muestra la información de los recursos necesarios para el funcionamiento de la aplicación. Su cometido es el de solicitar la concesión de los permisos de localización en caso de ser necesarios (API 23 o mayor) y la activación de la conectividad bluetooth si esta está desactivada.

Conexión

En la implementación de la vista de la conexión se ha utilizado el componente *RecyclerView* para el listado de los dispositivos.

Para establecer la conexión se ha implementado un servicio de Android. Este servicio está compuesto por dos hilos, el primero se encarga de establecer la conexión con el dispositivo y el segundo de la transferencia de datos. A este componente se le ha aplicado el componente **singleton** para poder obtener la misma instancia del objeto a lo largo de la ejecución de la aplicación.

La conexión se establece usando un UUID predefinido: **00001101-0000-1000-8000-00805F9B34FB**. Este se utiliza para conectarse a módulos serie bluetooth.

Puesto que la introducción se muestra una sola vez, antes de mostrar la vista de la conexión se realiza una comprobación de la disponibilidad de los recursos. En caso de faltar alguno, el fragmento se sustituye por otro que solicita el recurso que se necesita.

Calibración

Como se ha especificado en la sección 3.2.2, se ha utilizado el componente *SeekBar* para la introducción de valores de calibración.

Por detrás, utiliza las preferencias para guardar los valores introducidos por el usuario y poder usarlo posteriormente. También hace uso del servicio de bluetooth para enviar comandos de calibración.

Control y grabación

El principal componente usado en las vistas correspondientes a estas dos funcionalidades es el *JoystickView*. Como se ha comentado en la sección 4.2.1, la implementación de este componente está basado en [27].

El componente ha sido modificado para que tenga forma cuadrada en lugar de circular. Este cambio se ha realizado debido a que, en caso de usar un *joystick* circular, no se podría alcanzar el rango completo de ángulos de los servomotores (mover 90º en cada dirección). También se le han añadido unos ejes para facilitar la visibilidad del movimiento, puesto que dependiendo de eje sobre el que se realice el movimiento se moverá el pie o un lado de la cadera. Los movimientos sobre el eje X se transforman en movimientos del servo de la cadera mientras que los movimientos sobre el eje Y se transforman en movimientos del pie.

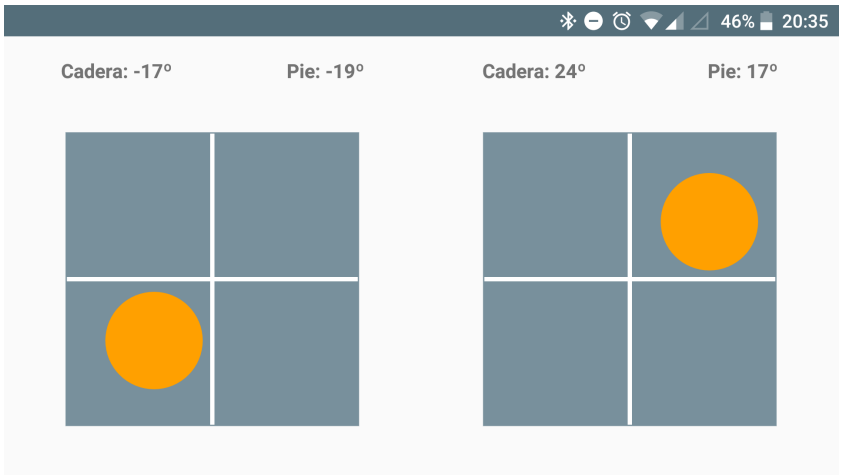

Figura 4.1: Joystick virtual

En la anterior imagen se puede observar el funcionamiento de los *joysticks* con los cuales se le ordena al robot mover el servo del lado izquierdo la cadera 17º a la izquierda, el del pie izquierdo 19º hacia abajo, el del lado derecho de la cadera 24º a la derecha y el del pie derecho 17º hacia arriba. El rango de los ángulos que se pueden alcanzar con los *joysticks* depende de la intensidad con la que se haya calibrado.

Para evitar la repetición de código, y puesto que se presenta la misma vista, en ambos se utiliza la misma clase *presentador*. La diferencia entre ambas funcionalidades radica

en la implementación de la interfaz que gestiona los cambios en la **UI** y en el uso del sistema de ficheros para generar la grabación.

La implementación de la interfaz consiste en la herencia del fragmento *Controller-Fragment*, y la sobrecarga de los métodos de la interfaz para adecuarlos a la grabación de comandos.

Grabaciones y reproducción

En estas dos últimas funcionalidades destaca el uso del sistema de ficheros, puesto que se realizan modificación de nombres, lectura y eliminación de ficheros.

Al igual que en la vista de conexión, se utiliza el componente *RecyclerView* para el listado de las grabaciones. Cada una de las grabaciones tiene la opción de ser renombrada o eliminada. En caso de la última, se utiliza el componente *SeekBar* para la notificación y anulación de la eliminación.

Para la reproducción de la grabación se utilizan *handlers* para el envío de comandos, los cuales son enviados dejando un tiempo entre comando y comando, correspondiente al tiempo que había pasado entre ellos en el momento de la grabación.

5

Pruebas, integración y resultados

Una vez se ha expuesto el desarrollo, se procede a detallar las pruebas que se han hecho durante el mismo. A lo largo del desarrollo se han hecho pruebas unitarias sobre el proyecto Android y pruebas de integración entre la aplicación móvil y el robot, teniendo éstas últimas más relevancia.

5.1. Pruebas unitarias

Como se ha comentado en capítulos anteriores, se ha usado el patrón **MVP** para facilitar la tarea de realizar pruebas de la aplicación. Al implementar el código de esta forma, se ha podido abstraer código, que se encapsula en las clases *presentador* de cada una de las funcionalidades. Cada una de estas clases actúa de intermediario entre las acciones que realiza el usuario y las consecuencias que estas tienen en la interfaz gráfica.

Con las pruebas unitarias solo se ha podido testear la correcta implementación de estas clases, por lo que no se cubre la totalidad del código generado.

Para realizar estas pruebas se han utilizado los *frameworks JUnit* y *Mockito* y la librería *Hamcrest*.

Se han realizado pruebas unitarias sobre 6 clases de las cuales se han obtenido los

siguientes resultados.

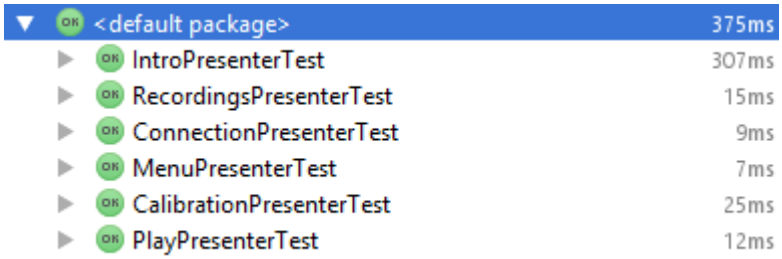

Figura 5.1: Pruebas unitarias

5.2. Pruebas de integración

Conforme se ha ido avanzando en el desarrollo, tanto del software del robot como el de la aplicación móvil, se han ido probando las distintas funcionalidades que se iban desarrollando. Los casos de pruebas que se han realizado son los siguientes:

1. Conexión con el robot.
2. Calibrado del robot.
3. Control del robot.
4. Reproducción de la grabación.
5. Perdida de la conexión bluetooth.

Los resultados de estas pruebas se pueden ver en la plataforma de contenido audiovisual *YouTube* [33].

5.2.1. Conexión con el robot

En el caso de la conexión tenemos dos posibilidades: conexión exitosa y conexión errónea.

Teniendo el robot encendido (conectado a una fuente de alimentación) se procede al establecimiento de la conexión obteniendo como resultado una conexión exitosa.

Con el robot apagado, se intenta establecer la conexión y al producirse un *timeout*, se notifica al usuario de que no se ha podido establecer la conexión.

5.2.2. Calibrado del robot

En este caso de prueba, con la opción de calibración abierta en el móvil, se procede a modificar los valores de los ángulos y la intensidad. Al enviar los valores, el robot refleja esos cambios.

5.2.3. Control del robot

Con la opción de control o grabación, se realizan movimientos con los *joysticks*. Consecuentemente, el robot mueve los servos en relación a los movimientos realizados en el móvil y a la calibración previamente hecha.

5.2.4. Reproducción de una grabación

En este caso de prueba, reproducimos una grabación y comprobamos que los movimientos que realiza el robot se corresponden a los que se hicieron en el momento de realizar la grabación. De la misma manera, podemos apreciar cómo se comporta el robot al aplicar las acciones de pausar, reanudar, parar y reproducir la grabación.

5.2.5. Perdida de conexión bluetooth

En caso de que se pierda la conexión, ya sea porque el robot se desplaza fuera del rango de alcance del bluetooth o se le desconecta de la fuente de alimentación, la aplicación actúa consecuentemente, redirigiendo a la vista de conexión para establecer nuevamente la conexión entre el robot y la aplicación.

5.3. Generación de patrones motores

A continuación, se muestra un ejemplo de lo que se puede conseguir con el uso de la aplicación y que se ha realizado para poner en práctica la generación de patrones motores.

Para este ejemplo se escogió una de las grabaciones que se realizó con la aplicación. Con los datos que esta contenía, se crearon gráficas, las cuales se analizaron para generar un nuevo movimiento en el que el robot anda hacia adelante y hacia atrás con un estilo que hemos llamado «andar de pato». Con este nuevo patrón, se implementó otro *sketch* para cargar el movimiento en el robot.

La siguiente gráfica se corresponde a la comparativa de movimientos que se realizaron sobre los servomotores de la cadera.

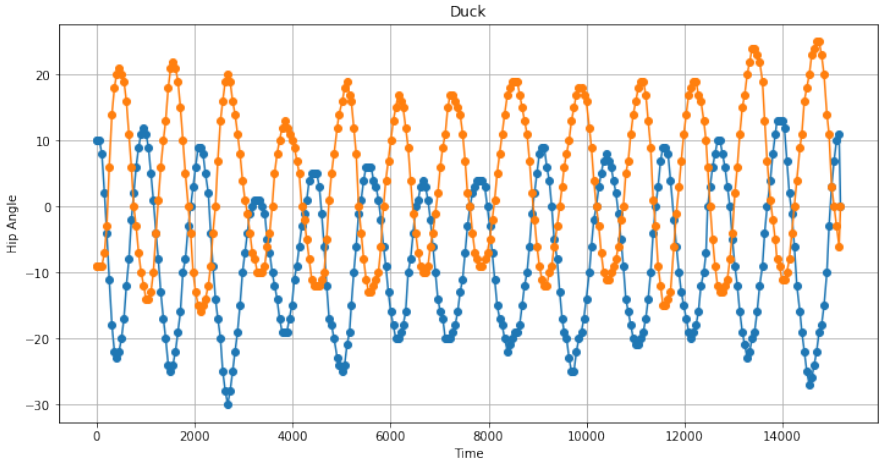

Figura 5.2: Servomotores de la cadera

En esta se puede observar que los movimientos que se realizan en ambos servos son completamente opuestos, por lo tanto, tienen un desfase de 180^{o} uno respecto del otro. Con el análisis de esta gráfica y junto con comparativas entre los demás servomotores, se consiguió crear el patrón que se nombró antes.

6

Conclusiones y trabajo futuro

Finalmente, se presentan las conclusiones sobre el proyecto realizado y sus resultados, al igual que las posibles líneas de trabajo futuro relacionadas con el mismo.

6.1. Conclusiones

Con la finalización de este proyecto, se puede apreciar que se han cumplido todos los objetivos que se marcaron en el comienzo.

El resultado final se compone de un robot, basado en la plataforma Arduino, y una aplicación móvil Android con la que se controla dicho robot. Esta aplicación proporciona versatilidad al uso del robot con el cual podremos hacer una amplia variedad de movimientos. Con un poco de habilidad y práctica se puede llegar a conseguir que el pequeño robot se mueva de formas que aún no nos imaginamos.

Además, al no usar funciones predefinidas para el movimiento del robot, controlarlo se convierte en un pequeño reto que te motiva a seguir intentando obtener nuevos movimientos.

6.2. Trabajo futuro

Como se ha dicho antes, se han conseguido los objetivos marcados en este proyecto, pero esto no quiere decir que de este no se puedan derivar trabajos para dar continuidad al mismo o aumentar su complejidad.

Este proyecto se enfoca de manera que el usuario ya posea cierta capacidad de razonamiento lógico. Teniendo esto en cuenta, se podría derivar este proyecto para abstraer más la generación de patrones motores y que esta se hiciera de manera automática, ampliando así la funcionalidad de la aplicación móvil.

Además, se podría aumentar la complejidad del montaje del robot, añadiendo nuevos componentes. Por ejemplo, se podría aumentar el número de servomotores utilizados para conseguir más grados de libertad y, al mismo tiempo, un movimiento más realista.

También se podría añadir sensores, como giroscopios o acelerómetros, que permitan determinar la posición del robot y ayudar a equilibrarlo.

Glosario de acrónimos

- **TIC**: Tecnologías de la información y la comunicación
- **ICTs**: Information and Communication Technologies
- **API**: Application Programming Interface
- **PWM**: Pulse-width modulation
- **MAC**: Media Access Control
- **IDE**: Integrated Development Environment
- **MVP**: Model-View-Presenter
- **RFCOMM** : Radio Frequency Communication
- **UUID** : Universally Unique Identifier
- **UI** : User Interface

Bibliografía

[1] Carlos Otto. Los robots ya están preparados para enseñarnos a programar. http://www.lavanguardia.com/tecnologia/20170618/423433107472/robots-ensenaran-hijos-programar.html, 2017.

[2] EFE. Los robots toman las aulas como nuevo recurso para la enseñanza. http://www.20minutos.es/noticia/2644370/0/robot-aula-educacion/nuevo-recurso/ensenanza/, 2016.

[3] Wikipedia. Robótica educativa. https://es.wikipedia.org/wiki/Rob

[4] Wikipedia. Programación visual. https://es.wikipedia.org/wiki/Programaci%C3%B3n_visual.

[5] Ranjit Kumar Barai, Mohd Razali Daud, Addie Irawan, and Kenzo Nonami. *Hydraulically Actuated Hexapod Robots.* Springer, 2013.

[6] B. Vanderborght. *Dynamic Stabilisation of the Biped Lucy Powered by Actuators with Controllable Stiffness.* Springer Tracts in Advanced Robotics. Springer Berlin Heidelberg, 2010.

[7] José Miguel García. Robótica educativa. la programación como parte de un proceso educativo. 2015.

[8] Nacho Palou. Robots educativos para iniciarse este verano en la robótica. http://tecnologia.elpais.com/tecnologia/2017/06/15/actualidad/1497515641_960941.html, 2017.

[9] Juan Jesús Velasco. Niños programadores: para qué sirve la enseñanza de programación en las escuelas. http://www.eldiario.es/turing/

Ninos-programadores-ensenanza-programacion-escuelas_0_293970921.html, 2014.

[10] BQ. Zowi. https://www.bq.com/es/zowi.

[11] El mundo. Bq presenta a zowi, el robot educativo. http://www.elmundo.es/tecnologia/2015/10/14/561e776022601da8758b4589.html.

[12] Javier Penalva. Probamos zowi, un robot con cerebro arduino que puede dar más de lo que aparenta. https://www.xataka.com/analisis/probamos-zowi-un-robot-con-cerebro-arduino-que-puede-dar-mas-de-lo-que-aparenta, 2016.

[13] Creatiev Commons. Attribution-sharealike 4.0 international. https://creativecommons.org/licenses/by-sa/4.0/.

[14] Javier Isabel. Zowi. https://github.com/JavierIH/zowi.

[15] Arduino. Introduction. https://www.arduino.cc/en/Guide/Introduction.

[16] Wiring. Wiring. http://wiring.org.co/.

[17] Wikipedia. Arduino. https://es.wikipedia.org/wiki/Arduino.

[18] Wikipedia. Servomotor. https://es.wikipedia.org/wiki/Servomotor.

[19] Area Tencología. Qué es un servomotor. http://www.areatecnologia.com/electricidad/servomotor.html.

[20] Panamahitek. Qué es y cómo funciona un servomotor. http://panamahitek.com/que-es-y-como-funciona-un-servomotor/.

[21] El Androide Libre. Qué es material design. https://elandroidelibre.elespanol.com/2014/11/que-es-material-design.html.

[22] Google. Material design. https://material.io/guidelines/.

[23] Google. Android testing codelab. https://codelabs.developers.google.com/codelabs/android-testing/.

[24] Oscar Arrivi. El patrón modelo-vista-presentador (mvp) a examen. http://theartoftheleftfoot.blogspot.com.es/2010/10/el-patron-modelo-vista-presentador-mvp.html, 2010.

[25] MSDN. El patrón singleton. https://msdn.microsoft.com/es-es/library/bb972272.aspx.

[26] Jake Wharton. Butter knife. http://jakewharton.github.io/butterknife/.

[27] Damien Brun. Virtual-joystick-android. https://github.com/controlwear/virtual-joystick-android.

[28] Apache. Apache license, version 2.0. http://www.apache.org/licenses/LICENSE-2.0.

[29] Android Developers. Permisos del sistema. https://developer.android.com/guide/topics/security/permissions.html.

[30] Android Developers. Cambios en android 6.0. https://developer.android.com/about/versions/marshmallow/android-6.0-changes.html.

[31] Android Developers. Uses feature element. https://developer.android.com/guide/topics/manifest/uses-feature-element.html.

[32] Android Developers. Servicios. https://developer.android.com/guide/components/services.html.

[33] Jonathan Yajamín. Resultados de las pruebas de integración. https://youtu.be/JrTEkV9J290, 2017.

A

Manual de instalación

Para poder utilizar el software desarrollado es necesario realizar dos procesos de instalación:

- Instalación del software en el robot.
- Instalación de la aplicación en el dispositivo móvil.

A.1. Instalación del código Arduino

Para la primera instalación es necesario el IDE de Arduino. En él se cargará el *sketch* que contiene el programa del robot, **Receiver.ino**. Con el entorno de desarrollo abierto, se conecta el robot al ordenador y se le introduce la siguiente configuración en la opción de *Herramientas*.

Clave	Valor
Placa	Arduino BT
Procesador	ATmega 328
Puerto	Puerto asignado al conectar el robot

Cuadro A.1: Configuración del IDE de Arduino

Una vez se haya configurado el entorno y con el robot conectado, se pulsa sobre el botón *Subir*.

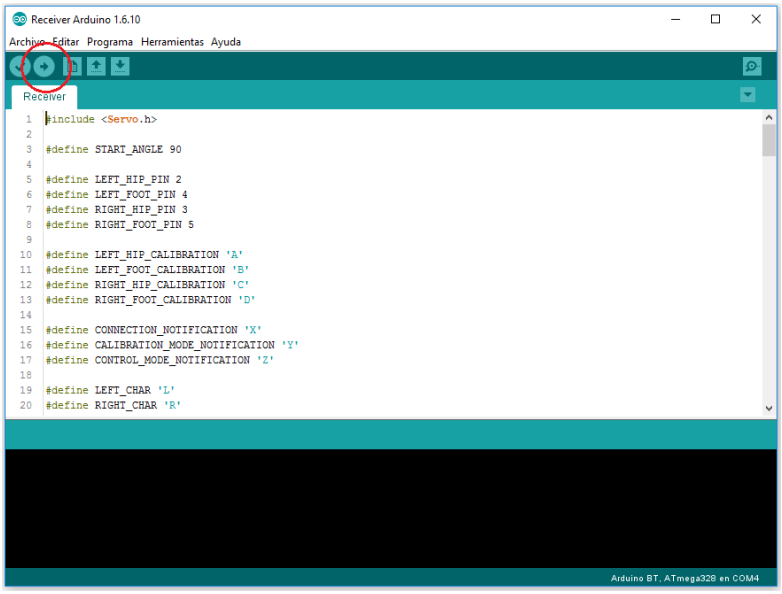

Figura A.1: IDE de Arduino

A.2. Instalación de la aplicación Android

La instalación de la aplicación se puede realizar por dos medios: usando el IDE de Android o mediante el fichero de instalación APK.

A.2.1. Entorno de desarrollo

Para la instalación de la aplicación con el entorno de desarrollo, se abre el proyecto y se pulsa sobre el botón *Ejecutar*.

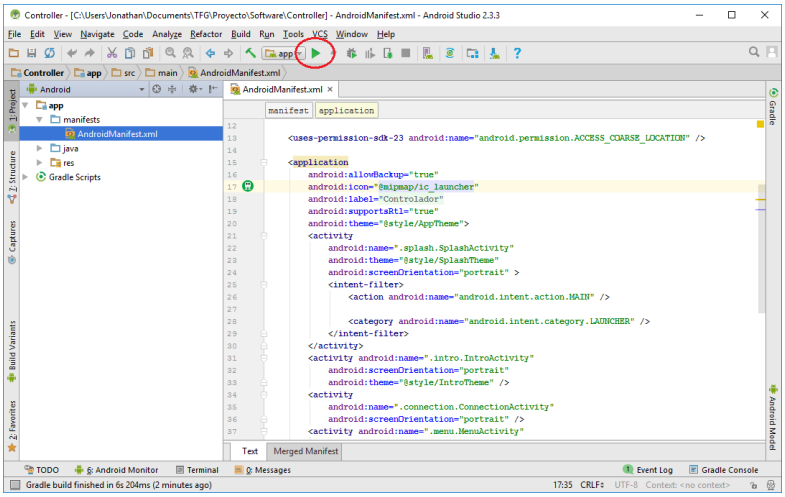

Figura A.2: IDE de Android

Después de pulsar el botón, aparecerá en pantalla una ventana donde se ha de escoger el dispositivo donde se quiere hacer la instalación. Al seleccionar el dispositivo y pulsar el botón *OK* se instalará la aplicación en el dispositivo móvil.

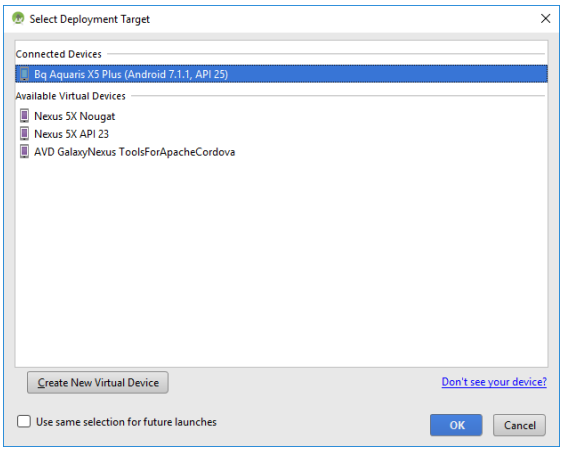

Figura A.3: Ventana de dispositivos

A.2.2. Fichero APK

La instalación mediante el fichero APK es más sencillo. Basta con abrir el explorador de ficheros para buscar el archivo APK y abrirlo pulsando una vez sobre él. A continuación, se muestra una pantalla de confirmación de instalación. Se pulsa el botón *Instalar*

y se termina el proceso.

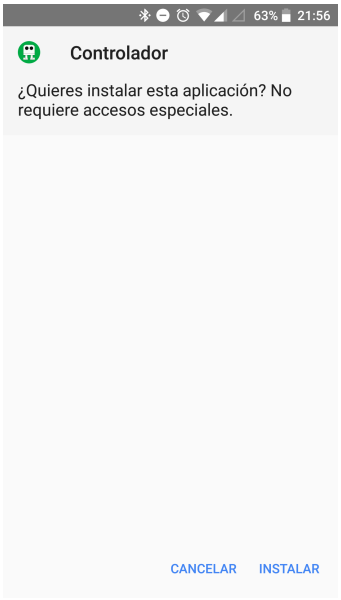

Figura A.4: Pantalla de instalación del dispositivo móvil

B

Manual del programador

En este anexo se muestran las secciones de código más relevantes.

Uno de los ficheros más relevantes es el que contiene el programa que se carga en el micro-controlador.

```
#include <Servo.h>

#define START_ANGLE 90

#define LEFT_HIP_PIN 2
#define LEFT_FOOT_PIN 4
#define RIGHT_HIP_PIN 3
#define RIGHT_FOOT_PIN 5

#define LEFT_HIP_CALIBRATION 'A'
#define LEFT_FOOT_CALIBRATION 'B'
#define RIGHT_HIP_CALIBRATION 'C'
#define RIGHT_FOOT_CALIBRATION 'D'

#define CONNECTION_NOTIFICATION 'X'
#define CALIBRATION_MODE_NOTIFICATION 'Y'
#define CONTROL_MODE_NOTIFICATION 'Z'

#define LEFT_CHAR 'L'
#define RIGHT_CHAR 'R'
#define SEPARATOR ','
```

```
#define END_OF_LINE ';'

#define LINE_FEED 10
#define CARRIAGE_RETURN 13

#define DEFAULT_OFFSET 0

enum ExecutionMode { Calibration, Control };
enum CalibrationAction { LeftHip, LeftFoot, RightHip, RightFoot };
enum Side { Left, Right };

String command = "";
char letter;
int letterValue;

int leftHipBaseAngle = START_ANGLE;
int leftFootBaseAngle = START_ANGLE;
int rightHipBaseAngle = START_ANGLE;
int rightFootBaseAngle = START_ANGLE;

int leftHipOffset = DEFAULT_OFFSET;
int leftFootOffset = DEFAULT_OFFSET;
int rightHipOffset = DEFAULT_OFFSET;
int rightFootOffset = DEFAULT_OFFSET;

int hipAngle;
int footAngle;

ExecutionMode executionMode;
CalibrationAction calibrationAction;
Side side;

Servo leftHipServo;
Servo leftFootServo;
Servo rightHipServo;
Servo rightFootServo;

void setup() {
    Serial.begin(19200);
    Serial.println("Robot receiver");

    leftHipServo.attach(LEFT_HIP_PIN);
    leftFootServo.attach(LEFT_FOOT_PIN);
    rightHipServo.attach(RIGHT_HIP_PIN);
    rightFootServo.attach(RIGHT_FOOT_PIN);
}

void loop() {
```

```
    while(Serial.available()){
        letter = Serial.read();
        letterValue = (unsigned int)letter;

        if(letter == CONNECTION_NOTIFICATION){
            Serial.println("Connection accomplished");
        }
        else if(letter == CALIBRATION_MODE_NOTIFICATION){
            Serial.println("CalibrationAction mode activated");
            executionMode = Calibration;
        }
        else if(letter == CONTROL_MODE_NOTIFICATION){
            Serial.println("Control mode activated");
            executionMode = Control;
        }
        else if(letterValue != LINE_FEED && letterValue !=
   CARRIAGE_RETURN){
            if(executionMode == Calibration)
                calibrateRobot();
            else if(executionMode == Control)
                moveRobot();
        }
    }
}

void calibrateRobot(){
    if(letter == LEFT_HIP_CALIBRATION)
        calibrationAction = LeftHip;
    else if(letter == LEFT_FOOT_CALIBRATION)
        calibrationAction = LeftFoot;
    else if(letter == RIGHT_HIP_CALIBRATION)
        calibrationAction = RightHip;
    else if(letter == RIGHT_FOOT_CALIBRATION)
        calibrationAction = RightFoot;
    else if(letter == SEPARATOR){
        switch(calibrationAction){
            case LeftHip:
                leftHipOffset = atoi(command.c_str());
                leftHipBaseAngle = START_ANGLE + leftHipOffset;
                break;

            case LeftFoot:
                leftFootOffset = atoi(command.c_str());
                leftFootBaseAngle = START_ANGLE + leftFootOffset;
                break;

            case RightHip:
                rightHipOffset = atoi(command.c_str());
                rightHipBaseAngle = START_ANGLE + rightHipOffset;
```

```
                break;
        }

        command = "";
    }
    else if(letter == END_OF_LINE){
    rightFootOffset = atoi(command.c_str());
    rightFootBaseAngle = START_ANGLE + rightFootOffset;

        leftHipServo.write(leftHipBaseAngle);
        leftFootServo.write(leftFootBaseAngle);
        rightHipServo.write(rightHipBaseAngle);
        rightFootServo.write(rightFootBaseAngle);

        command = "";
    }
    else if((letter >= '0' && letter <= '9') || letter == '-' ||
   letter == '+' || letter == '.')
        command += letter;
    else{
        Serial.println("Unknown char in calibration mode: " + letter)
   ;
        command = "";
    }
}

void moveRobot(){
    if(letter == LEFT_CHAR)
        side = Left;
    else if(letter == RIGHT_CHAR)
        side = Right;
    else if(letter == SEPARATOR){
        hipAngle = atoi(command.c_str());
        command = "";
    }
    else if(letter == END_OF_LINE){
        footAngle = atoi(command.c_str());

        if(side == Left){
            leftHipServo.write(leftHipBaseAngle + hipAngle);
            leftFootServo.write(leftFootBaseAngle + footAngle);
        }
        else if(side == Right){
            rightHipServo.write(rightHipBaseAngle + hipAngle);
            rightFootServo.write(rightFootBaseAngle + footAngle);
        }

        command = "";
    }
```

```
    else if((letter >= '0' && letter <= '9') || letter == '-' ||
    letter == '+')
        command += letter;
    else{
        Serial.println("Unknown char in control mode: " + letter);
        command = "";
    }
}
```

Receiver.ino

A continuación, se va a mostrar las clases que se han implementado para realizar la conexión bluetooth. El primero se corresponde a una clase donde se ha encapsulado las acciones que se realizan con el adaptador bluetooth. El segundo contiene la implementación del servicio bluetooth que se usa para el establecimiento de la conexión, así como de la transferencia de datos.

```
package es.uam.eps.tfg.controller.util;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;

import java.util.Set;

/**
 * @author Jonathan Yajamín
 */

public class BluetoothManager {
    /* Interfaces */
    public interface OnDiscoveryListener {
        void onStartDiscovery();
        void onStopDiscovery();
        void onDeviceDiscovery(BluetoothDevice bluetoothDevice);
        void onDeviceConnected();
        void onDeviceDisconnected();
    }

    public static final int REQUEST_ENABLE = 1;

    private BluetoothAdapter bluetoothAdapter;
    private BroadcastReceiver oldReceiver;
```

```
    private BroadcastReceiver receiver;
    private IntentFilter filter;
    private Context context;

    public BluetoothManager(){
        bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
    }

    public static boolean isSupported(){
        return BluetoothAdapter.getDefaultAdapter() != null;
    }

    public static boolean isEnabled(){
        return BluetoothAdapter.getDefaultAdapter().isEnabled();
    }

    public static void enable(Activity activity){
        Intent intent = new Intent(BluetoothAdapter.
ACTION_REQUEST_ENABLE);
        activity.startActivityForResult(intent, REQUEST_ENABLE);
    }

    public Set<BluetoothDevice> getPairedDevices(){
        return bluetoothAdapter.getBondedDevices();
    }

    public void searchAvailableDevices(){
        if(bluetoothAdapter.isDiscovering())
            bluetoothAdapter.cancelDiscovery();

        bluetoothAdapter.startDiscovery();
    }

    public void setDiscoveryFilters(Context context, String[] actions
){
        this.context = context;
        filter = new IntentFilter();
        for (String action : actions)
            filter.addAction(action);

        registerReceiver();
    }

    public void setDiscoveryFilters(Context context, String[] actions
, BroadcastReceiver receiver){
        this.context = context;
        this.receiver = receiver;
        filter = new IntentFilter();
        for (String action : actions)
```

```
            filter.addAction(action);

        registerReceiver();
    }

    public void setOnDiscoveryListener(final OnDiscoveryListener
listener){
        oldReceiver = receiver;
        receiver = new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {
                String action = intent.getAction();

                switch (action){
                    case BluetoothAdapter.ACTION_DISCOVERY_STARTED:
                        listener.onStartDiscovery();
                        break;

                    case BluetoothAdapter.ACTION_DISCOVERY_FINISHED:
                        listener.onStopDiscovery();
                        break;

                    case BluetoothDevice.ACTION_FOUND:
                        BluetoothDevice device = intent.
getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                        listener.onDeviceDiscovery(device);
                        break;

                    case BluetoothDevice.ACTION_ACL_CONNECTED:
                        listener.onDeviceConnected();
                        break;

                    case BluetoothDevice.ACTION_ACL_DISCONNECTED:
                        listener.onDeviceDisconnected();
                        break;
                }
            }
        };
    }

    public void updateOnDiscoveryListener(OnDiscoveryListener
listener){
        setOnDiscoveryListener(listener);
        registerReceiver();
    }

    public void removeOnDiscoveryListener(){
        if(receiver != null)
            context.unregisterReceiver(receiver);
```

```
        oldReceiver = null;
        receiver = null;
    }

    private void registerReceiver(){
        if(oldReceiver != null)
            context.unregisterReceiver(oldReceiver);

        context.registerReceiver(receiver, filter);
    }
}
```

BluetoothManager.java

```
package es.uam.eps.tfg.controller.util;

import android.app.Service;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Context;
import android.content.Intent;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.support.annotation.Nullable;
import android.util.Log;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.UUID;

/**
 * @author Jonathan Yajamín
 */

public class BluetoothService extends Service {

    public interface OnConnectionErrorListener{
        void onConnectionError();
    }

    private BluetoothAdapter adapter;
    private UUID moduleUuid = UUID.fromString("
    00001101-0000-1000-8000-00805F9B34FB");

    private Context context;
```

```
    private ConnectingThread connectingThread;
    private ConnectedThread connectedThread;
    private boolean stopThread;

    private Handler handler;

    private static BluetoothService instance;
    public static OnConnectionErrorListener errorListener;

    @Override
    public void onCreate() {
        super.onCreate();
        instance = this;
        stopThread = false;
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId)
{
        handler = new Handler(){

            @Override
            public void handleMessage(Message msg) {
                if(msg.what == 0){
                    String message = msg.obj.toString();
                    Log.d("Message", message);
                }
            }
        };

        context = this.getBaseContext();
        String deviceAddress = intent.getStringExtra(Constants.
MAC_KEY);
        adapter = BluetoothAdapter.getDefaultAdapter();
        BluetoothDevice device = adapter.getRemoteDevice(
deviceAddress);
        connectingThread = new ConnectingThread(device);
        connectingThread.start();

        return super.onStartCommand(intent, flags, startId);
    }

    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    @Override
```

```
    public void onDestroy() {
        super.onDestroy();

        stopThread = true;
        handler.removeCallbacksAndMessages(null);
        if(connectingThread != null) connectingThread.closeSocket();
        if(connectedThread != null) connectedThread.closeStreams();
    }

    public void send(String message){
        connectedThread.write(message);
    }

    public static BluetoothService getInstance(){
        return instance;
    }

    private class ConnectingThread extends Thread {
        private BluetoothSocket socket;

        public ConnectingThread(BluetoothDevice device){
            try{
                socket = device.createRfcommSocketToServiceRecord(
moduleUuid);
            } catch (IOException exception){
                stopSelf();
            }
        }

        @Override
        public void run() {
            super.run();

            adapter.cancelDiscovery();
            try{
                socket.connect();

                connectedThread = new ConnectedThread(socket);
                connectedThread.start();

                connectedThread.write(CommandManager.
CONNECTION_NOTIFICATION);
            } catch (IOException exception) {
                errorListener.onConnectionError();
                try {
                    socket.close();
                    stopSelf();
                } catch (IOException innerException) {
                    stopSelf();
```

```
                }
            }
        }

        public void closeSocket(){
            try {
                socket.close();
            } catch (IOException innerException) {
                stopSelf();
            }
        }
    }

    private class ConnectedThread extends Thread {
        private InputStream inputStream;
        private OutputStream outputStream;

        public ConnectedThread(BluetoothSocket socket){
            try {
                inputStream = socket.getInputStream();
                outputStream = socket.getOutputStream();
            } catch (IOException exception) {
                stopSelf();
            }
        }

        @Override
        public void run() {
            byte[] buffer = new byte[256];
            int bytes;

            while(true && !stopThread){
                try {
                    bytes = inputStream.read(buffer);
                    String message = new String(buffer, 0, bytes);
                    handler.obtainMessage(0, bytes, -1, message).
    sendToTarget();
                } catch (IOException e) {
                    stopSelf();
                    break;
                }
            }
        }

        public void write(String input){
            byte[] buffer = input.getBytes();
            try {
                outputStream.write(buffer);
            } catch (IOException e) {
```

```
                stopSelf();
            }
        }

        public void closeStreams(){
            try {
                inputStream.close();
                outputStream.close();
            } catch (IOException exception) {
                stopSelf();
            }
        }
    }
}
```

BluetoothService.java

C

Componentes hardware

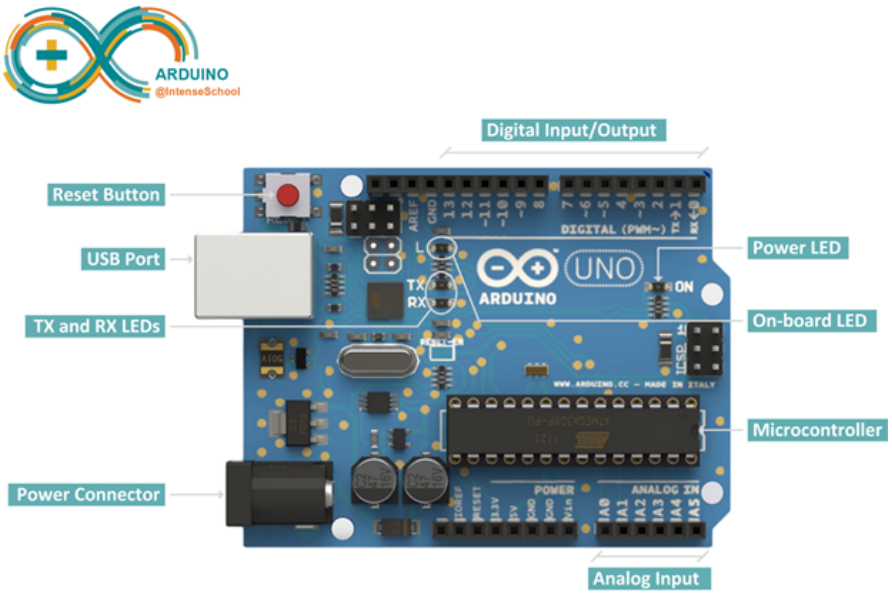

Figura C.1: Arduino Uno

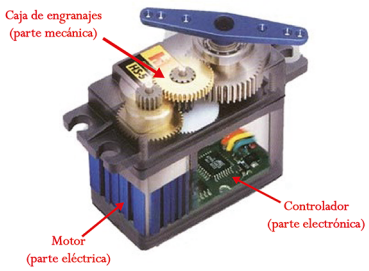

Figura C.2: Composición de un servomotor

Voltaje positivo	Tierra (ground)	Señal de control

Figura C.3: Colores de los terminales de conexión de un servomotor

Figura C.4: Servomotor Futaba S3003

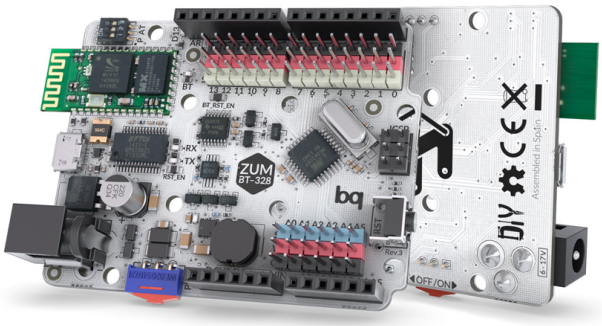

Figura C.5: Placa BQ ZUM BT 238

D

Flujo de la aplicación

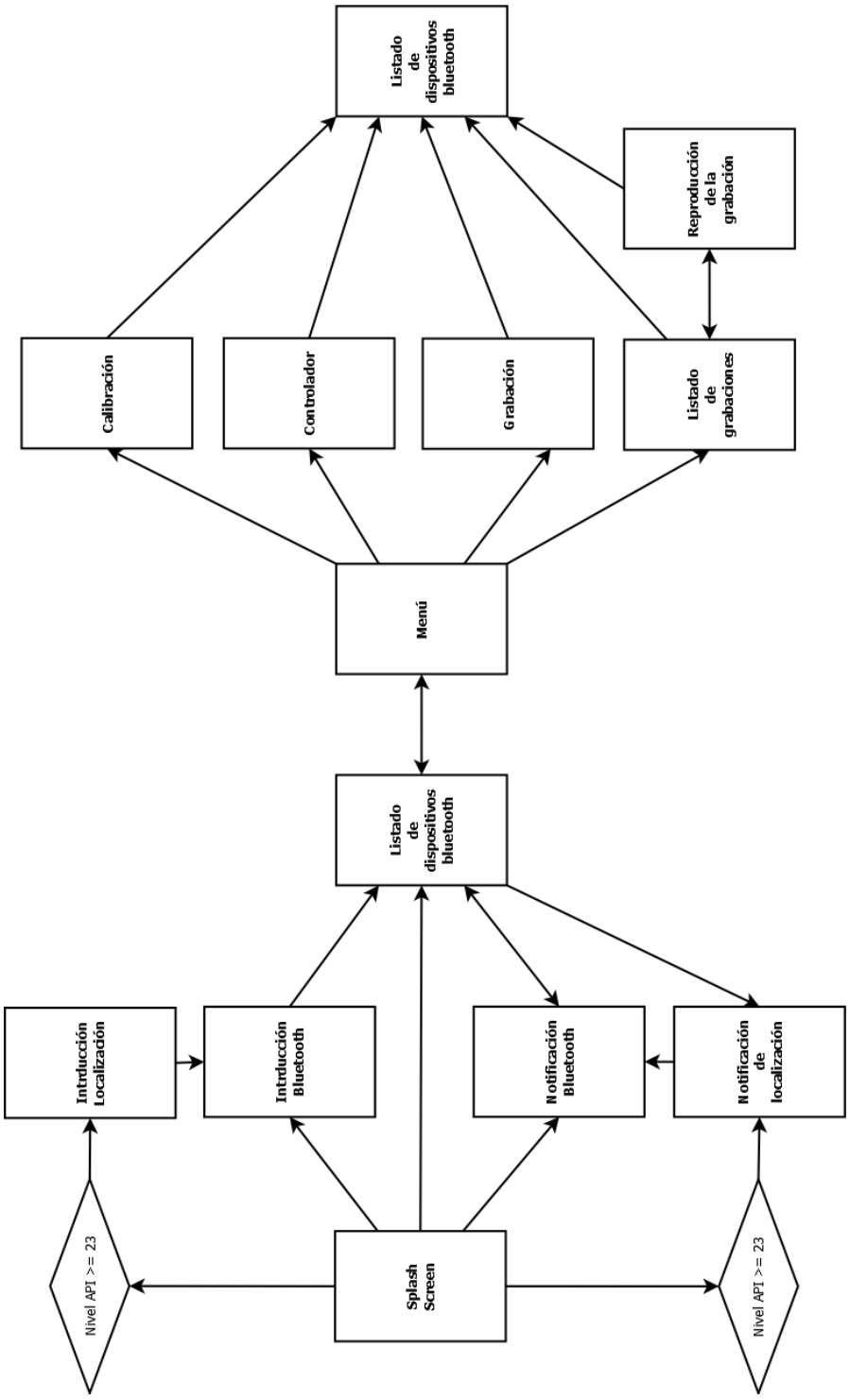

Figura D.1: Flujo de la aplicación

E

Vistas de la aplicación

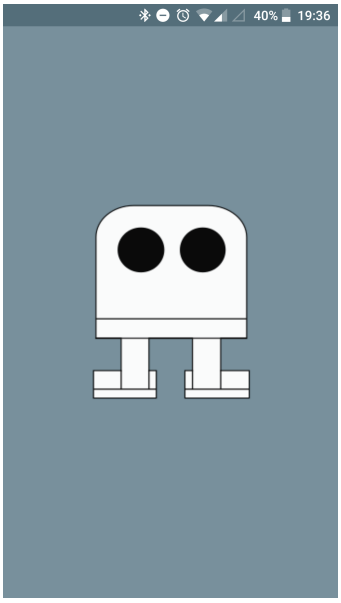

Figura E.1: Splash screen

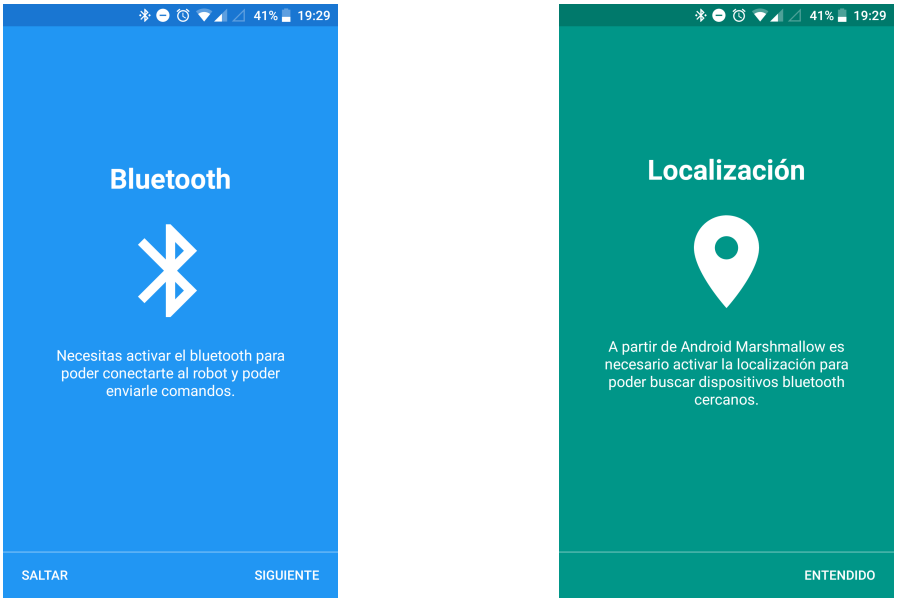

Figura E.2: Vistas de la introducción

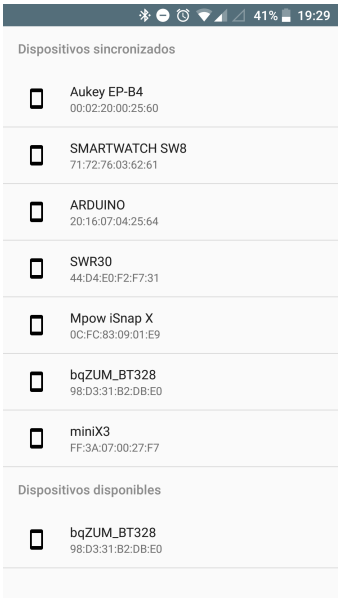

Figura E.3: Vista de los dispositivos bluetooth

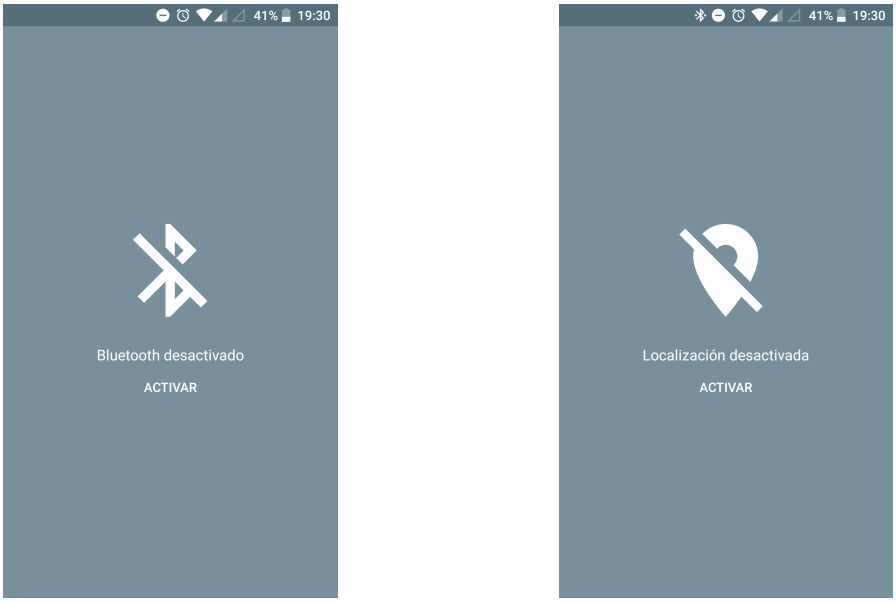

Figura E.4: Vistas de las notificaciones

Figura E.5: Vista del menú

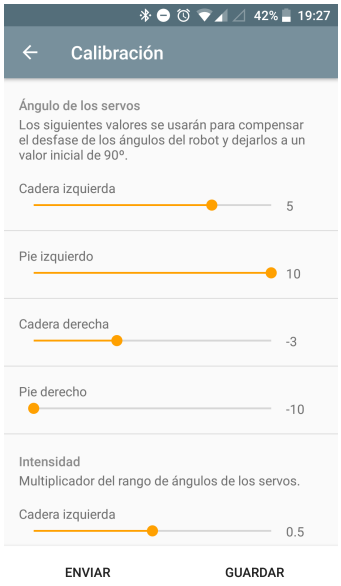

Figura E.6: Vista de la calibración

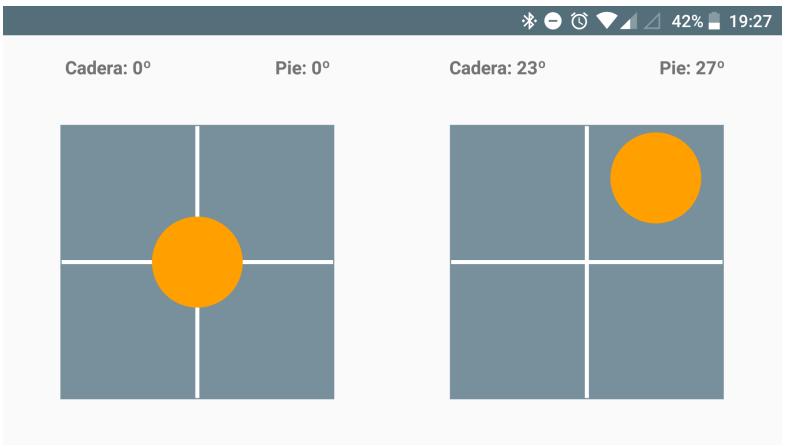

Figura E.7: Vista de control y grabación

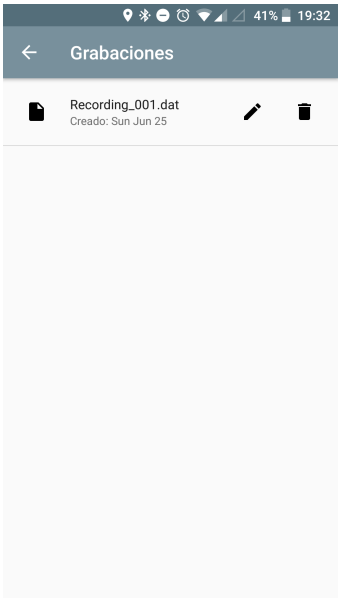

Figura E.8: Vista de las grabaciones

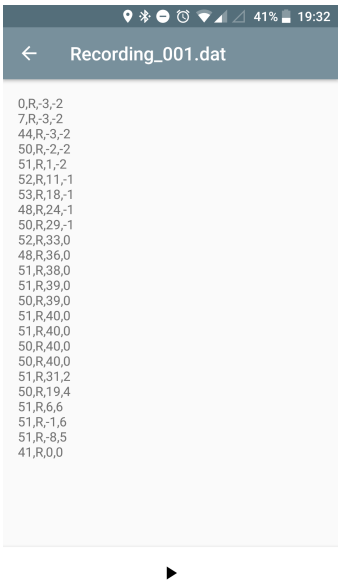

Figura E.9: Vista de reproducción de grabación